UNITED STATES PATENT APPLICATION

for

MECHANISM FOR HANDLING RACE CONDITIONS IN FRC-ENABLED PROCESSORS

Inventors:

Andrew J. Honcharik
3842 S. Sage Court
Chandler, AZ 85248
Citizen of The United States

Steven J. Tu
3527 East Windmere Drive
Phoenix, AZ 85048
Citizen of The United States

Hang T. Nguyen
8613 S. Dorsey Lane
Tempe, AZ 85284
Citizen of The United States

Sujat Jamil
1828 Enfield Way
Chandler, AZ 85248
Citizen of Bangladesh

Merrell Quinn
15827 S. 27th Way
Phoenix, AZ 85048
Citizen of The United States

File No.: 042390.P12103

"Express Mail" mailing label number: EL 867 651 169 US
Date of Deposit: **December 31, 2001**
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Commissioner for Patents, Washington, D. C. 20231

Maureen R. Pettibone
(Typed or printed name of person mailing paper or fee)

Maureen R. Pettibone
(Signature of person mailing paper or fee)

12/31/01
(Date signed)

MECHANISM FOR HANDLING RACE CONDITIONS IN FRC-ENABLED PROCESSORS

Background of the Invention

[0001] Technical Field The present invention relates to microprocessors and, in particular, to mechanisms for handling errors in FRC-enabled processors.

[0002] Background Art. Servers and other high-end computer system are designed to provide high levels of reliability and availability. Soft errors pose a major challenge to both of these properties. Soft errors result from bit-flipping collisions between high-energy particles, e.g. alpha particles, and charge storing nodes. They are prevalent in storage arrays, such as caches, TLBs, and the like, which include large numbers of charge storing nodes. Rates of occurrence of soft errors (soft error rates or SERs) will only increase as device geometry decreases and device densities increase.

[0003] Highly reliable systems include safeguards to detect and manage soft errors, before they lead to silent, e.g. undetected, data corruption (SDC). However, to the extent error detection/handling mechanisms take a system away from its normal operations, the system's availability is reduced.

[0004] One well-known mechanism for detecting soft errors is functional redundancy checking (FRC). An FRC-enabled processor includes replicated instruction execution cores on which the same instruction code is run. Depending on the particular embodiment, each replicated execution core may include one or more caches, register files and supporting resources in addition to the basic execution units (integer, floating point, load store, etc.). FRC-hardware compares results generated by each core, and if a discrepancy is detected, the FRC system jumps to an error-handling routine. Since FRC errors indicate only that the execution cores disagree on

a result, the error handling routine typically unwinds execution to the last know point of reliable data and resets the processor. This reset mechanism is relatively time consuming.

[0005] The point(s) at which results from different execution cores are compared represent the FRC-boundary for the system. Errors that are not detected at the FRC boundary can lead to SDC, but even managing detected FRC errors can create its own problems. As noted above, time spent by the system handling FRC-errors takes the system away from its normal operation, reducing system availability.

[0006] FRC is only one mechanism for handling soft errors, and for random logic, it is the primary mechanism. Array structures present a different picture. Array structures typically include parity and/or ECC hardware to detect soft errors. ECC hardware can even correct single bit errors on the fly. Further, unlike FRC errors, which only indicate that one of the execution cores generated an error, parity/ECC errors are clearly tied to one of the execution cores by the logic that detects them.

[0007] If a parity/ECC error in an execution core is detected before the corrupted data reaches the FRC boundary, the execution core that has the good, i.e. uncorrupted, data can be identified and used to recover from the error. If the corrupted data reaches the FRC boundary before the underlying parity/ECC error is detected, an FRC error will be generated, and the system will be unavailable while the FRC error is addressed. There is thus a race between the logic that processes the corrupted data towards the FRC boundary and the logic that detects its corrupted state. This race can have a significant impact on system availability, due to the longer latency of the reset mechanism triggered by an FRC-error (data mismatch) relative to the recovery mechanism triggered by an underlying parity/ECC error.

[0008] The present invention address these and other problems associated with error handling in FRC-enabled processors.

Brief Description of the Drawings

[0009] The present invention may be understood with reference to the following drawings, in which like elements are indicated by like numbers. These drawings are provided to illustrate selected embodiments of the present invention and are not intended to limit the scope of the invention.

[0010] Fig. 1 is a block diagram of an FRC-enabled processor that includes dual execution cores and FRC-detection and handling logic.

[0011] Fig. 2 is a block diagram of one embodiment of an FRC-enabled processor that implements an error-detection and handling system in accordance with the present invention.

[0012] Figs. 3A and 3B are block diagrams representing the states of one embodiment of the FRC-checker of Fig. 2 at two different times.

[0013] Figs. 4A and 4B are block diagrams representing the states of another embodiment of FRC-checker of Fig. 2 at two different times.

[0014] Fig. 5 is a flowchart representing one embodiment of a method 500 for handling race conditions in FRC-enabled systems in accordance with the present invention.

[0015] Fig. 6 is a block diagram of one embodiment of a computer system that implements an error handling system in accordance with the present invention.

Detailed Description of the Invention

[0016] The following discussion sets forth numerous specific details to provide a thorough understanding of the invention. However, those of ordinary skill in the art, having the benefit of this disclosure, will appreciate that the invention may be practiced without these specific details. In addition, various well-known methods, procedures, components, and circuits have not been described in detail in order to focus attention on the features of the present invention.

[0017] Fig. 1 is a block diagram representing an FRC-enabled processor 110. Processor 110 includes first and second execution cores 120(a), 120(b) (generically, execution core 120), an FRC checker 130, an error detector 140, a recovery unit 150, and reset unit 160. A portion of the FRC-boundary is indicated by dashed line 104. For purposes of illustration, recovery unit 150 and reset unit 170 are shown as part of processor 110, but portions of these units may be implemented as firmware or software modules that are located off of the processor die.

[0018] Each execution core 120 includes a data pipeline 124 and an error pipeline 128 that feed into FRC checker 130 and error detector 140, respectively. Data pipeline 124 represents the logic that operates on various types of data as it moves through processor 110 toward FRC checker 130. The data represented by data pipeline 124 may include result operands, status flags, addresses, instructions and the like that are generated and staged through processor 110 during code execution. Error pipeline 128 represents the logic that operates on various types of data to detect errors in the data and provide appropriate error signals to error detector 140. For example, the error signals may be one or more bits (flags) representing the parity or ECC status of data retrieved from various storage arrays (not shown) of processor 110. Soft errors in these arrays may appear as parity or ECC error flags when the corrupted data is accessed.

[0019] If an error reaches error detector 140 from either core 120, recovery unit 160 is activated to implement a recovery routine. Recovery can be effected with relatively low latency by hardware, software, firmware or some combination of these. For example, there is an extremely small probability that the same data is corrupted in both execution cores 120, which leaves an uncorrupted copy of the data available to restore data integrity to processor 110. However, an FRC error will be triggered if the corrupted and uncorrupted data is allowed to reach FRC checker 130 before recovery unit 160 is activated. Since FRC errors are not recoverable, reset module 170 will unwind the machine state of processor 110 and reset it, if FRC checker 130 signals an FRC error before the underlying parity/ECC error is detected.

[0020] Not all FRC-errors are traceable to underlying parity/ECC or other correctible soft errors. For those that are, it is faster for error detector 140 to address the underlying soft error than for FRC-checker to address the FRC error that results when the corrupted data reaches FRC boundary 104. As noted above, the reset process has a significantly longer latency than the recovery process, and it is to be avoided if the error can be corrected by recovery unit 150. For this reason, FRC checker 130 is temporarily disabled if error detector 140 detects an error in either error pipeline 128.

[0021] Data pipelines 120 operate in lock step during normal FRC mode, but data pipeline 124 and error pipeline 128 may operate relatively independently. For example, ECC hardware is relatively complex and consequently, relatively slow, especially for 2-bit errors. A flag signaling such an error may reach error detector 140 before, after, at the same time as the data with which it is associated reaches FRC checker 130. This flexibility is generally beneficial. For example, it allows data to be used speculatively before its error status is determined. Since soft errors are relatively rare, and error pipeline 124 is generally as fast as data pipeline 128, this flexibility is

net positive. As long as the error flag arrives at error detector 140 in time to disable FRC checker 130 before it receives the corrupted data, the relatively low latency recover routine is engaged.

[0022] As noted above, races between data pipeline 124 and error pipeline 128 can reduce system availability in certain cases if not addressed. If an error flag indicating a soft error reaches detector 140 too late to disable FRC checker 130 from processing the associated data, a recoverable error becomes an FRC error, and the longer latency reset routine is engaged. The longer latency reset routine reduces the availability of system 100 for normal operation.

[0023] Fig. 2 is a block diagram of one embodiment of an FRC-enabled processor 210 that handles race conditions effectively. Processor 210 includes first and second execution cores 220(a), 220(b) (generically, execution core 220), an FRC checker 230, an error detector 240, a recovery unit 250, and reset unit 260. An FRC-boundary for processor 210 is indicated by dashed line 204. As noted above, portions of recovery unit 250 and reset unit 260 may be implemented as firmware or software modules that are located off of the processor die.

[0024] Each execution core 220 includes a data pipeline 224 and an error pipeline 228 that feeds into FRC checker 230 and error detector 140, respectively. Execution cores 220(a), 220(b) and their components operate in substantially the same manner as described above for execution cores 120(a), 120(b).

[0025] Error detector 240 monitors error pipelines 228(a), 228(b). If a signal or flag indicates an error in one of execution cores 220, error detector 240 disables FRC checker 230 and activates recovery unit 250. As discussed below in greater detail, FRC checker 240 provides buffering that allows error detector 230 to activate recovery unit 250, even if it detects the error after FRC checker 230 detects a mismatch between the corresponding corrupted and uncorrupted

data. Provided the error is detected within a countdown interval initiated by an FRC mismatch, the FRC mismatch is ascribed to the error, and processor 210 activates a recovery routine through recovery unit 250 rather than a slower reset routine through reset unit 260.

[0026] Towards this end, the disclosed embodiment of FRC checker 230 includes a compare/buffer unit 234, a queue 236 and a timer unit 238. Compare/buffer unit 234 receives data from execution cores 220(a), 220(b), compares the data, and temporarily stores data from at least one execution core 220 along with a status flag to indicate if the comparison yielded a match. If the data matches, the status flag is set to indicate the match, and the data is buffered to queue 236 for continued processing by resources outside FRC boundary 204. As discussed below, hold times for data in compare/buffer unit 234 depend on the number of entries it provides, which may depend on the countdown interval selected.

[0027] If the data does not match, the status flag is set to indicate the mismatch and timer unit 238 is triggered to begin a countdown interval. Compare/buffer unit 234 may continue to receive, compare and store data from execution cores 220(a), 220(b) during the countdown interval. Similarly, error detector 240 may continue to monitor error pipelines 228(a), 228(b). If error detector 240 receives an error flag before the timeout interval expires, it disables FRC checker 230 from receiving additional data from execution cores 220(a), 220(b) and triggers recovery unit 250 to implement the recovery routine for the entry tagged as a mismatch in compare/buffer unit 234.

[0028] Operation of FRC checker 230 and execution cores 220 during the countdown interval that follows an FRC mismatch may vary with different embodiments of the invention. For example, the stall behavior of execution cores 230 and the source of uncorrupted data may

differ depending on whether results from one or both execution cores 220 are stored by compare/buffer 234.

[0029] Figs. 3A and 3B are block diagrams of one embodiment of a processor 310 in accordance with the present invention at two different times. Contents of data pipeline 324 and error pipeline 328 are labeled to indicate the relative times at which they are processed through execution core 320. The labeled data blocks are not necessarily valid or fully developed until they exit pipeline 324. For purposes of the following discussion, execution core 320(a) operates as the master and execution core 320(b) operates as the slave, when processor 310 is in FRC mode.

[0030] The disclosed embodiment of compare/buffer 334 includes a buffer, a comparator 370 and buffer entries 380(1)-380(3) (generically, buffer entries 380). Comparator 370 compares data from execution cores 320(a) and 320(b) and generates a match status signal according to the comparison. Comparator 370 also triggers a stall signal to one of execution cores 320 if the status signal indicates a mismatch. Each buffer entry 380 includes a data field 384 to store the data from master execution core 320(a) and a status field 388 to store the status determined by comparator 370.

[0031] For the instant reflected in Fig. 3A, data_0 has been accepted and forwarded to queue 338 for further processing. Here, "accepted" means that the data_0 blocks provided by master and slave execution cores 320(a), 320(b), respectively, matched, and no FRC error is indicated by comparator 370. Data_1 through data_3 are accepted and currently stored in buffer entries 380(1)-380(3), awaiting transfer to queue 338. Data_4 through data_8 are currently propagating through data pipelines 224(a), 224(b) of master and slave execution cores, 220(a), 220(b), respectively. An error associated with data_4 in data pipeline 324(a) (indicated by dashed line)

is propagating through error pipeline 328(a) approximately two clock cycles behind data_4. For example, the error may represent a 2-bit soft error detected by ECC hardware. The detection hardware for 2-bit ECC errors is relatively complex, and may contribute to the delay indicated between data_4 and its error flag.

[0032] Fig. 3B shows processor 310 at a later instant in which data_3 from execution core 320(a) has been forwarded to queue 336, data_4 - data_6 have been transferred to buffer 380 and their match status determined. In particular, comparator 370 has signaled a mismatch (reject) for data_4, which was corrupted by a soft error, and its uncorrupted counterpart from execution core 320(b). Comparator 370 has also triggered the stall signal to slave execution core 320(b) and triggered timer unit 338 to begin the countdown interval. For the disclosed embodiment, stalling one of data pipelines 324 ensures that an uncorrupted copy of the data is preserved for recovery unit 360, in either the stalled pipeline or compare/buffer unit 334.

[0033] During the countdown interval, the disclosed embodiment of FRC checker 330 continues to receive data from master execution core 320(a). For the illustrated example, data_5 and data_6 have been by stored during this interval with rejected status, since corresponding elements from stalled slave execution core 320(b) are unavailable for comparison.

[0034] During the countdown interval, error detector 340 also continues to track flags from error pipelines 328. If the error flag associated with data_4 reaches error detector 340 before timer unit 338 signals the end of the countdown interval, error detector 340 disables FRC checker 330 and activates recovery unit 360. If timer unit 338 detects the end of the countdown interval before error detector 340 detects the error flag, reset unit 370 is activated.

[0035] Figs. 4A and 4B are block diagrams of another embodiment of a processor 410 in accordance with the present invention at two different times. For processor 410, the disclosed

embodiment of compare/buffer 434 includes a comparator 470 and buffer entries 480(1)-480(3) (generically, buffer entries 480). Comparator 470 compares data from execution cores 420(a) and 420(b) and generates a match status signal according to the comparison. Each buffer entry 480 includes data fields 384 and 388 to store data from execution cores 420(a) and 420(b), respectively, and a status field 486 to store the status determined by comparator 470.

[0036] For system 400, data blocks from both pipelines are transferred to FRC checker 430. Consequently, no stall signal is needed to preserve uncorrupted data in one of execution cores 420 following detection of an FRC mismatch by comparator 470. The uncorrupted data may be provided from the appropriate entry of buffer 480.

[0037] Timer unit 438 initiates a countdown interval in response an FRC mismatch detected by comparator 470. FRC checker 430 and error detector 440 continue to process data and flags, respectively, during the countdown interval. If an error reaches error detector 440 before the countdown interval expires, error detector 440 disables FRC checker 430 and activates recovery unit 460. If the countdown interval expires first, FRC checker 430 activates reset unit 470 to reset system 400.

[0038] Fig. 5 is a flowchart representing one embodiment of a method 500 in accordance with the present invention for handling race conditions. Method 500 reflects the interactions between concurrent non-recoverable (FRC) error checking 504 and recoverable (non-FRC) error checking 508 processes.

[0039] Recoverable error checking process 508 monitors 580 flags/signals from an FRC-enabled processor. The monitored flags may be generated by parity checking hardware, ECC hardware or other units capable of detecting recoverable errors attributable to, e.g. soft errors within the FRC boundary of the processor. The flags may be provided to an error checker

through pathways having various latencies. If an error flag is detected 590, a signal is asserted 554 to the FRC checker and a recovery algorithm is activated 558. The signal asserted to the FRC checker prevents it from triggering a non-recoverable error.

[0040] Non-recoverable error checking process 504 operates concurrently with recoverable error checking process 508. Data from the redundant cores of an FRC-enabled processor is tracked 510 and compared 520. If the data matches 520, no FRC-error is indicated and process 504 continues monitoring. If a mismatch is detected 520, a potential FRC-error is indicated.

[0041] To determine whether the potential FRC-error is an artifact of a race condition, a countdown interval is activated 530 and data tracking continues 540. If a recoverable (e.g. non-FRC) error is detected 550 during the countdown interval, process 504 relinquishes control to the recovery algorithm (558). If the countdown interval expires 560 before a non-FRC error is signaled, a reset algorithm is executed 570. If the countdown interval has not yet expired 560, process 504 continues monitoring 540 data until the countdown interval expires (560) or a non-FRC error is indicated 550.

[0042] Fig. 6 is a block diagram representing one embodiment of a computer system 600 in which the present invention is implemented. The disclosed embodiment of system 600 includes a processor 610, chipset 670, main memory 680, non-volatile memory 690 and peripheral device(s) 694. Chipset 670 manages communications among processor 610, main memory 680, non-volatile memory 690 and peripheral devices 694.

[0043] Processor 610 includes first and second execution cores 620(a) and 620(b), respectively (generically, execution cores 620). Each execution core includes execution resources 624 and a bus cluster 628. Execution resources 624 may include, for example, one or more integer, floating point, load/store, and branch execution units, as well register files and

cache(s) to supply them with data (e.g instructions, operands, addresses). Bus cluster 628 represents logic for managing transactions to a cache 640 that is shared by execution cores 620(a) and 620(b), as well as to a front side bus 660, for those transactions that miss in shared cache 640.

[0044] Error check units 630(a), 630(b) (generically, error check units 630) compare results from execution cores 620 before they are transferred to shared resources like cache 640 and FSB 660. They thus form part of the FRC boundary of processor 610. Each error check unit 630 includes an FRC checker 230 and error detector 240 (Fig. 2) to provide FRC and non-FRC error checking/detection, respectively, for execution cores 620. Error check units 630 may also include portions of recovery unit 240 and reset unit 260 (Fig. 2). For one embodiment of system 600, a recovery routine 692 and a reset routine 694 may be stored in non-volatile memory 690 and images of these routines may be loaded in main memory 680 for execution. For this embodiment, recovery unit 240 and reset unit 260 may include pointers to recovery routine 692 and reset routine 694, respectively (or their images in main memory 680).

[0045] The disclosed embodiment of system 600 also includes interrupt controllers 670(a) and 670(b) (generically, interrupt controller(s) 670) to process interrupts for execution cores 620(a) and 620(b), respectively. Each interrupt controller 670 is shown having first and second components 674 and 678, respectively, to accommodate the different clock domains in which interrupt controller 670 may operate. For example, FSB 660 typically operates at a different frequency than processor 610. Consequently, components of processor 610 that interact directly with FSB 660 typically operate in its clock domain, which is indicated as area 664 on processor 610.

[0046] The disclosed embodiment of interrupt controller 670 also includes an FRC-boundary-like component in the form of XOR 672. XOR 672 signals an FRC error if it detects a mismatch between outgoing signals, e.g. interrupt responses, of components 674(a) and 674(b) from execution cores 620(a) and 620(b). Errors attributable to interrupt controllers 670 may still arise, however, from soft errors in components 678(a), 678(b) in FSB clock domain 664. These error may be detected by the discrepancies they introduce between the subsequent operations of execution cores 620(a) and 620(b).

[0047] For the disclosed embodiment of system 600, a common snoop block 662 processes snoop transactions to and from execution cores 620(a) and 620(b). XOR 666 provides FRC-checking on snoop responses from execution cores 620(a), 620(b) and signals an error if a mismatch is detected.

[0048] There has thus been disclosed a mechanism for handling race conditions between recoverable and non-recoverable errors in an FRC-enabled processor. An FRC checker includes a buffer to temporarily store data representing results from redundant execution cores along with status bits to indicate if the results matched. An error detector monitors error hardware to determine if any of the results may have been subject to a recoverable error, such as a parity or ECC error. If the FRC checker detects a mismatch, a potential FRC error is indicated, and a timer is activated to provide the error detector with additional time to identify a recoverable error from which the FRC error originated. If a recoverable error is detected before the countdown interval expires, a recovery mechanism is triggered. If the interval expires before a recoverable error is detected, a reset mechanism is triggered. For either recovery or reset mechanisms, the FRC-enabled processor may switch temporarily into a non-FRC mode to allow one core to execute the reset/recovery code.

[0049] The disclosed embodiments have been provided to illustrate various features of the present invention. Persons skilled in the art of processor design, having the benefit of this disclosure, will recognize variations and modifications of the disclosed embodiments, which none the less fall within the spirit and scope of the appended claims.